A Real Coded Genetic Algorithm for Data Partitioning and Scheduling in Networks with Arbitrary Processor Release Time

S. Suresh[1], V. Mani[1], S. N. Omkar[1], and H. J. Kim[2]

[1] Department of Aerospace Engineering,
Indian Institute of Science, Bangalore, India
[2] Department of Control and Instrumentation Engineering
Kangwon National University, Chunchon 200-701, Korea

Abstract. The problem of scheduling divisible loads in distributed computing systems, in presence of processor release time is considered. The objective is to find the optimal sequence of load distribution and the optimal load fractions assigned to each processor in the system such that the processing time of the entire processing load is a minimum. This is a difficult combinatorial optimization problem and hence genetic algorithms approach is presented for its solution.

1 Introduction

One of the primary issues in the area of parallel and distributed computing is how to partition and schedule a divisible processing load among the available processors in the network/system such that the processing time of the entire load is a minimum. In the case of computation-intensive tasks, the divisible processing load consists of large number of data points, that must be processed by the same algorithms/programs that are resident in all the processors in the network. Partitioning and scheduling computation-intensive tasks incorporating the communication delays (in sending the load fractions of the data to processors) is commonly referred to as divisible load scheduling. The objective is to find the optimal sequence of load distribution and the optimal load fractions assigned to each processor in the system such that the processing time of the entire processing load is a minimum. The research on the problem of scheduling divisible loads in distributed computing systems started in 1988 [1] and has generated considerable amount of interest among researchers and many more results are available in [2, 3]. Recent results in this area are available in [4].
Divisible load scheduling problem will be more difficult, when practical issues like processor release time, finite buffer conditions and start-up time are considered. It is shown in [5] and [6], that this problem is NP hard when the buffer constraints and start-up delays in communication are included. A study on computational complexity of divisible load scheduling problem is presented in [6]. The effect of communication latencies (start-up time delays in communication) are studied in [7–11] using single-round and multi-round load distribution strategies. The

problem of scheduling divisible loads in presence of processor release times is considered in [12–14]. In these studies, it is assumed that the processors in the network are busy with some other computation process. The time at which the processors are ready to start the computation of its load fraction (of the divisible load) is called "release time" of the processors.
The release time of processors in the network, affect the partitioning and scheduling the divisible load. In [12], scheduling divisible loads in bus network (homogeneous processors) with identical and non-identical release time are considered. The heuristic scheduling algorithms for identical and non-identical release time are derived based on multi-instalment scheduling technique presented based on the release time and communication time of the complete load. In [14], heuristic strategies for identical and non-identical release time are presented for divisible load scheduling in linear network. In these studies the problem of obtaining the optimal sequence of load distribution by the root processor is not considered.
In this paper, the divisible load scheduling problem with arbitrary processor release time in single level tree network is considered. When the processors in the network have arbitrary release time, it is difficult to obtain a closed-form expression for optimal size of load fractions. Hence, for a network with arbitrary processor release times, there are two important problems: (i) For a given sequence of load distribution, how to obtain the load fractions assigned to the processors, such that the processing time of the entire processing load is a minimum, and (ii) For a given network, how to obtain the optimal sequence of load distribution.
In this paper, Problem(i), of obtaining the processing time and the load fractions assigned to the processors, for a given sequence of load distribution is solved using a real coded hybrid genetic algorithm. Problem (ii), of obtaining the optimal sequence of load distribution is a combinatorial optimization problem. For a single-level tree network with m child processors there are $m!$ sequences of load distribution are possible. Optimal sequence of load distribution is the sequence for which the processing time is a minimum. We use genetic algorithm to obtain the optimal sequence of load distribution. The genetic algorithm for obtaining the optimal sequence of load distribution uses the results of real-coded genetic algorithm used to solve problem(i). To the best of our knowledge, this is the first attempt to solve this problem of scheduling divisible with processor release times.

2 Definitions and Problem Formulation

Consider a single-level tree network with $(m+1)$ processors as shown in Figure 1. The child processors in the network denoted as p_1, p_2, $\cdots$, p_m are connected to the root processor (p_0) via communication links l_1, l_2, $\cdots$ l_m. The divisible load originates at the root processor (p_0) and the root processor divides the load into $(m+1)$ fractions $(\alpha_0, \alpha_1, \cdots, \alpha_m)$ and keeps the part α_0 for itself to process/compute and distributes the load fractions $(\alpha_1,\alpha_2, \cdots, \alpha_m)$ to other m processors in the sequence $(p_1, p_2, \cdots, p_m)$ one after another.

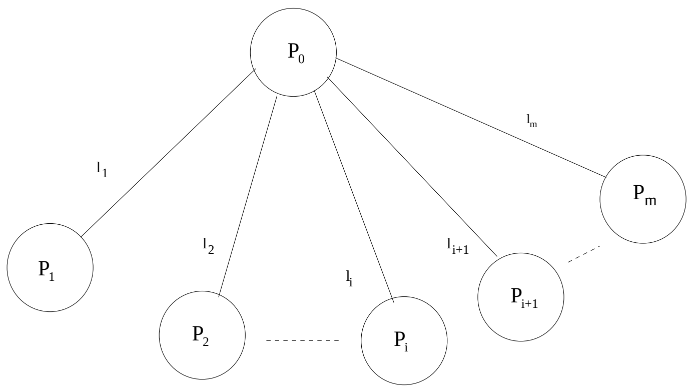

Fig. 1. Single-level tree network

The child processors in the network may not be available for computations process immediately after the load fractions are assigned to it. This will introduce a delay in starting the computation process of the load fraction. This delay is commonly referred as processor release time. The release time is different for different child processors. The objective here is to obtain the processing time and load fractions assigned to the processors. In this paper, we follow the standard notations and definitions used in divisible load scheduling literature.
Definitions:

- **Load distribution:** This is denoted as α, and is defined as an $(m+1)$-tuple $(\alpha_0,\ \alpha_1,\ \alpha_2,\ \cdots,\ \alpha_m)$ such that $0 < \alpha_i \leq 1$, and $\sum_{i=0}^{m} \alpha_i = 1$. The equation $\sum_{i=0}^{m} \alpha_i = 1$ is the normalization equation, and the space of all possible load distribution is denoted as Γ.
- **Finish time:** This is denoted as T_i and is defined as the time difference between the instant at which the i^{th} processor stops computing and the time instant at which the root processor initiates the load distribution process.
- **Processing time:** This is the time at which the entire load is processed. This is given by the maximum of the finish time of all processors; i.e.,$T = max\{T_i\}$ $i = 0, 1, \cdots, m$, where T_i is the finish time of processor p_i.
- **Release time:** The release time of a child processor p_i is the time instant at which the child processor is available to start the computation of its load fraction of the divisible load.

Notations:

- α_i: The load fraction assigned to the processor p_i.
- w_i: The ratio of the time taken by processor p_i, to compute a given load, to the time taken by a standard processor, to compute the same load;
- z_i: The ratio of the time taken by communication link l_i, to communicate a given load, to the time taken by a standard link, to communicate the load.

– T_{cp}: The time taken by a standard processor to process a unit load;
– T_{cm}: The time taken by a standard link to communicate a unit load.
– b_i: Release time for processor p_i.

Based on these notations, we can see that $\alpha_i w_i T_{cp}$ is the time to process the load fraction α_i of the total processing load by the processor p_i. In the same way, $\alpha_i z_i T_{cm}$ is the time to communicate the load fraction α_i of the total processing load over the link l_i to the processor p_i. We can see that both $\alpha_i w_i T_{cp}$ and $\alpha_i z_i T_{cm}$ are in units of time. In divisible load scheduling literature, timing di-

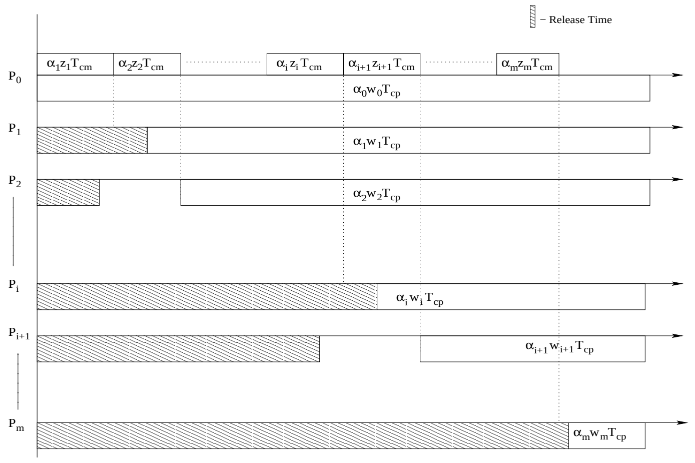

Fig. 2. Timing Diagram For Single Level Tree Network With Release Time

agram is the usual way of representing the load distribution process. In this diagram the communication process is shown above the time axis, the computation process is shown below the time axis and the release time is shown as shaded region below time axis. The timing diagram for the single-level tree network with arbitrary processor release time is shown in Figure. 2. From timing diagram, we can write the finish time (T_0) for the root processor (p_0) as

$$T_0 = \alpha_0 w_0 T_{cp} \tag{1}$$

Now, we find the finish time for any child processor p_i. The time taken by the root processor to distribute the load fraction (α_i) to the child processor p_i is $\sum_{j=1}^{i} \alpha_j z_j T_{cm}$. Let b_i be the release time of the child processor p_i then the child processor starts the computation process at max $\left(b_i, \sum_{j=1}^{i} \alpha_j z_j T_{cm}\right)$. Hence, the finish time (T_i) of any child processor p_i is

$$T_i = \max\left(b_i, \sum_{j=1}^{i} \alpha_j z_j T_{cm}\right) + \alpha_j w_j T_{cp} \quad i = 1, 2, \cdots, m \tag{2}$$

The above equation is valid, if the load fraction assigned to the child processor is greater than zero (the child processors participate in the load distribution process) otherwise $T_i = 0$.
From the timing diagram, the processing time (T) of the entire processing load is

$$T = \max(T_i, \forall\ i = 0, 1, \cdots, m) \tag{3}$$

Obtaining the processing time (T), and the load fractions (α_i) for the divisible load scheduling problem with arbitrary processor release time is difficult. Hence, in this paper, we propose a real coded genetic algorithm approach to solve the above scheduling problem.

3 Real-Coded Genetic Algorithm

The Genetic Algorithm (GA) is perhaps the most well-known of all evolution based search techniques. The genetic algorithm is a probabilistic technique that uses a population of solutions rather than a single solution at a time [15–17]. In these studies, the search space solutions are coded using binary alphabet. For optimization problems floating point representation of solution in the search space outperform binary representations because they are more consistent, more precise and lead to faster convergence. This fact is discussed in [18]. Genetic algorithms using real number representation for solutions are called real-coded genetic algorithms. More details about how genetic algorithms work for a given problem can be found in literature [15–18].
A good representation scheme for solution is important in a GA and it should clearly define meaningful crossover, mutation and other problem-specific operators such that minimal computational effort is involved in these procedures. To meet these requirements, we propose real coded hybrid genetic algorithm approach to solve the divisible load scheduling problem with arbitrary processor release time. The real coded approach seems adequate when tackling problems of parameters with variables in continuous domain [19, 20]. A detailed study on effect of hybrid crossovers for real coded genetic algorithm is presented in [21, 22].

3.1 Real-coded Genetic Algorithm for Divisible Load Scheduling Problem

In this paper, a hybrid real coded genetic algorithm is presented to solve the divisible scheduling problem with arbitrary processor release time. In real coded GA, solution (chromosome) is represented as an array of real numbers. The chromosome representation and genetic operators are defined such that it satisfies the normalization equation.

String Representation The string representation is the process of encoding a solution to the problem. Each string in a population represent possible solution to the scheduling problem. For our problem, the string consists of an array of real numbers. The values of real number represents the load fractions assigned to the processors in the network. The length of the string is $m + 1$, the number of processors in the system. A valid string is the one in which the total load fractions (sum of all real numbers) is equal to one.
In case of four ($m = 4$) processor system, a string will represent the load fractions $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ assigned to the processors p_0, p_1, p_2, p_3 and p_4 respectively. For example, a valid string $\{0.2, 0.2, 0.2, 0.2, 0.2\}$, in our problem represents the load fraction assigned to the processors ($\alpha_0 = 0.2$, $\alpha_1 = 0.2$, $\alpha_2 = 0.2$, $\alpha_3 = 0.2$ and $\alpha_4 = 0.2$) in the network. The sum of load fractions assigned to the processors is equal to 1. In general, for an m-processor system, the length of the string is equal to $m + 1$.

Population Initialization Genetic algorithms search from a population of solution points instead of a single solution point. The initial population size, and the method of population initialization will affect the rate of convergence of the solution. For our problem the initial population of solutions is selected in the following manner.

- Equal allocation: The value of load fraction (α_j) in the solution is $\frac{1}{M+1}$, for $j = 1, ..., M + 1$.
- Random allocation: Generate $M + 1$ random numbers. These random numbers are normalized such that the sum is equal to one. This is the value of α_j in the solution, for $j = 1, ..., M + 1$.
- Zero allocation: Select a solution using equal or random allocation. Any one element in the selected solution is assigned zero and its value is equally allocated to other elements.
- Proportional allocation: The value of load fraction α_j in the solution is $\alpha_j = \frac{w_j}{\sum_{i=1}^{M} w_i}$.

Selection function *Normalized Geometric Ranking Method*: The solutions (population) are arranged in descending order of their fitness value. Let q be the selection probability for selecting best solution and r_j be the rank of jth solution in the partially ordered set. The probability of solution j being selected using normalized geometric ranking method is

$$s_j = q^{'} (1 - q)^{r_j - 1} \tag{4}$$

where $q^{'} = \frac{q}{1-(1-q)^N}$ and N is the population size. The details of the normalized geometric ranking method can be found in [23].

Genetic operators Genetic operators used in genetic algorithms are analogous to those which occur in the natural world: reproduction (crossover, or recombination); mutation.

Crossover Operator is a primary operator in GA. The role of crossover operator is to recombine information from the two selected solutions to produce better solutions. The crossover operator improves the diversity of the solution vector. Four different crossover operators used in our divisible load scheduling problem are Two-point crossover (TPX), Simple crossover (SCX), Uniform crossover (UCX) and Averaging crossover (ACX). These crossover operators are described in [24].

Hybrid Crossover: We have used four types of crossover operators. The performance of these operators in terms of convergence to optimal solution depends on the problem. One type of crossover operator which performs well for one problem may not perform well for another problem. Hence many research works are carried out to study the effect of combining crossover operators in a genetic algorithm [25–28] for a given problem. Hybrid crossovers are a simple way of combining different crossover operators. The hybrid crossover operators use different kinds of crossover operators to produce diverse offsprings from the same parents. The hybrid crossover operator presented in this study generates eight offsprings for each pair of parents by SPX, TPX, UCX and ACX crossover operators. The most promising offsprings of the eight substitute their parents in the population.

Mutation Operator The mutation operator alters one solution to produce a new solution. The mutation operator is needed to ensure diversity in the population, and to overcome the premature convergence and local minima problems. Mutation operators used in this study are Swap mutation (SM) and Random Zero Mutation (RZM) are described in [24].

Fitness Function: The objective in our scheduling problem is to determine the load fractions assigned to the processors such that the processing time of the entire processing load is a minimum. The calculation of fitness function is easy. The string gives the load fractions α_0, α_1, $\cdots$, α_m assigned to the processors in the network. Once the load fractions are given, the finish time of all processors can be easily obtained. For example, the finish time T_i of processor p_i is $\max\left(b_i, \sum_{j=1}^{i} \alpha_j z_j T_{cm}\right) + \alpha_i w_i T_{cp}$. If the value of α_i is zero for any processor p_i, then the finish time of that processor T_i is zero. The processing time of the entire processing load T is $\max(T_i, \forall\ i = 0, 1, \cdots, m)$. Since, the genetic algorithm maximizes the fitness function, the fitness is defined as negative of the processing time (T).

$$F = -T \tag{5}$$

Termination Criteria In genetic algorithm, the evolution process continues until a termination criterion is satisfied. The maximum number of generations is the most widely used termination criterion and is used in our simulation studies.

3.2 Genetic Algorithm

Step 1. Select population of size N using initialization methods described earlier.
Step 2. Calculate the fitness of the solutions in the population using the equation (3).
Step 3. Select the solutions from the population using normalized geometric ranking method, for genetic operations.
Step 4. Perform different types of crossover and mutation on the selected solutions (parents). Select the N best solutions using elitist model.
Step 5. Repeat the Step 2-4 until the termination criteria is satisfied.

4 Numerical Example

We have successfully implemented and tested the real coded hybrid genetic algorithm approach for divisible load scheduling problems in tree network with arbitrary processor release time. The convergence of the genetic algorithm depends on population size (N), selection probability (S_c), crossover rate (p_c) and mutation rate (p_m). In our simulations the numerical values used are: $N = 30$, $S_c = 0.08$, $P_c = 0.8$, $p_m = 0.2$, and T_{cm}, T_{cp} are 1.0 .
Let us consider a single-level tree network with eight child processors ($m = 8$) attached to the root processor p_0. The computation and communication speed parameters, and processor release time are given in Table. 1. The root processor (p_0) distributes the load fractions to the child processors in the following sequence p_1, p_2, $\cdots$, p_8. The result obtained from real coded hybrid genetic algorithm is presented in Table. 1. The processing time of the entire processing load is 0.29717

Table 1. System Parameters and Results for Numerical Example 1.

Sequence	Comp. speed parameter	Comm. speed parameter	Release Time	Load Fraction
p_0	$w_0 = 1.1$	-	-	$\alpha_0 = 0.2702$
p_1	$w_1 = 1.5$	$z_1 = 0.4$	$b_1 = 0.15$	$\alpha_1 = 0.0981$
p_2	$w_2 = 1.4$	$z_2 = 0.3$	$b_2 = 0.1$	$\alpha_2 = 0.1408$
p_3	$w_3 = 1.3$	$z_3 = 0.2$	$b_3 = 0.50$	$\alpha_3 = 0$
p_4	$w_4 = 1.2$	$z_4 = 0.1$	$b_4 = 0.2$	$\alpha_4 = 0.0810$
p_5	$w_5 = 1.1$	$z_5 = 0.35$	$b_5 = 0.05$	$\alpha_5 = 0.1431$
p_6	$w_6 = 1.2$	$z_6 = 0.1$	$b_6 = 0.25$	$\alpha_6 = 0.0393$
p_7	$w_7 = 1.0$	$z_7 = 0.05$	$b_7 = 0.1$	$\alpha_7 = 0.1462$
p_8	$w_8 = 1.65$	$z_8 = 0.15$	$b_8 = 0.12$	$\alpha_8 = 0.0812$

In this numerical example, the processor-link pair (p_3,l_3) is removed from the network by assigning zero load fractions. The processors p_1, p_2, p_4, p_5, p_6, p_7 and p_8 receives their load fractions at times 0.0392, 0.0815, 0.0896, 0.1397, 0.1436

0.1509 and 0.1631. Thus, the processors p_1, p_2, p_4 and p_6 will start the computation process from the release time where as the processors p_5, p_7 and p_8 will be idle until their load fractions are received.

5 Optimal Sequence of Load Distribution

Sequence of load distribution is the order in which the child processors are activated in divisible load scheduling problem. In the earlier section, the sequence (activation order) of load distribution by the root processor is $\{p_1, p_2, \cdots, p_m\}$. For a system with m child processors there are $m!$ sequences of load distribution are possible. Optimal sequence of load distribution is the sequence of load distribution for which the processing time is a minimum.

First we will show that the problem of finding the optimal sequence of load distribution, is similar to the well-known Travelling Salesman Problem (TSP) studied in operations research literature. TSP is very easy to understand; a travelling salesman, must visit every city exactly once, in a given area and return to the starting city. The cost of travel between all cities are known. The travelling salesman to plan a sequence (or order) of visit to the cities, such that the cost of travel is a minimum. Let the number of cities be m. Any single permutation of m cities is a sequence (solution) to the problem. The number of sequences possible are $m!$. A genetic algorithm approach to TSP is well discussed in [18]. We now show the similarities between finding the optimal sequence of load distribution and Travelling Salesman Problem (TSP).

In TSP a solution is represented as string of integers representing the sequence in which the cities are visited. For example, the string $\{4\ 3\ 5\ 1\ 2\}$ represents the tour as $c_4 \longrightarrow c_3 \longrightarrow c_5 \longrightarrow c_1 \longrightarrow c_2$, where c_i is the city i. In our problem, the string $\{4\ 3\ 5\ 1\ 2\}$ represents the sequence of load distribution by the root processor to the child processors is $\{p_4\ p_3\ p_5\ p_1\ p_2\}$. So we can see the solution representation in our problem is the same as solution representation problem in TSP [18]. Hence, for our problem we have used the genetic algorithm approach given for TSP given in [18]. From genetic algorithm point of view, the only difference between TSP and divisible load scheduling problem is the fitness function. The fitness function is the cost of travel in TSP for the given sequence but the fitness function in our problem is the processing time for a given sequence of load distribution.

Fitness Function: The fitness function is based on the processing time of the entire processing load. Here for a given sequence, fitness function is the solution of real-coded genetic algorithm described in the earlier section. The objective in our problem is processing time minimization. Hence, in our case the fitness function is $(-T)$

$$F \;=\; -T \tag{6}$$

In order to determine the fitness (F), for any given sequence, the processing time (T), is obtained by solving the problem (i) using real-coded genetic algorithm methodology given in the earlier section. The optimal or best sequence of load

distribution can be found by using the genetic algorithm approach given for TSP given in [18], with the above fitness function.
Optimal Sequence for Numerical Example. 1: The optimal sequence of load distribution (by root processor p_0) obtained using the above genetic algorithm is $\{p_7\ p_5\ p_2\ p_8\ p_1\ p_6\ p_4\}$. The processing time for this sequence is $T = 0.2781$. The load fractions assigned to processors are: $\alpha_0 = 0.25286$, $\alpha_7 = 0.17814$, $\alpha_5 = 0.18568$, $\alpha_2 = 0.12015$, $\alpha_8 = 0.093447$, $\alpha_1 = 0.081151$, $\alpha_6 = 0.023453$, $\alpha_4 = 0.06512$. The processor p_3 is assigned a zero load fraction. Processor p_3 is assigned a zero load fraction because the release time ($b_3 = 0.5$) is high.

6 Conclusions

The problem of scheduling divisible loads in distributed computing system, in presence of processor release time is considered. In this situation, there are two important problems: (i) For a given sequence of load distribution, how to obtain the load fractions assigned to the processors, such that the processing time of the entire processing load is a minimum, and (ii) For a given network, how to obtain the optimal sequence of load distribution. A real-coded genetic algorithm is presented for the solution of problem (i). It is shown that problem (ii), of obtaining the optimal sequence of load distribution is a combinatorial optimization problem similar to Travelling Salesman Problem. For a single-level tree network with m child processors there are $m!$ sequences of load distribution are possible. Optimal sequence of load distribution is the sequence for which the processing time is a minimum. We use another genetic algorithm to obtain the optimal sequence of load distribution. The genetic algorithm for obtaining the optimal sequence of load distribution uses the real-coded genetic algorithm used to solve problem (i).

Acknowledgement: This work was in part supported by MSRC-ITRC under the auspices of Ministry of Information and Communication, Korea.

References

1. Y. C. Cheng and T. G. Robertazzi, "Distributed computation with communication delay", *IEEE Trans. Aerospace and Electronic Systems*, vol. 24, pp. 700-712, 1988.
2. V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, Los Alamitos, California, IEEE CS Press, 1996.
3. Special Issue on: Divisible Load Scheduling, *Cluster Computing*, Vol.6, Jan 2003.
4. http://www.ee.sunysb.edu/tom/dlt.html
5. M. Drozdowski and M. Wolniewicz, "On the complexity of divisible job scheduling with limited memory buffers", *Technical Report, R-001/2001*, Institute of Computing Science, Poznan University of Technology, 2001.
6. O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Independent and divisible task scheduling on heterogeneous star-shaped platforms with limited memory", *Research Report Number 2004-22*, LIP, ENS, Lyon, France, 2004.

7. O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible work loads on heterogeneous systems", *Proceedings of the International Parallel and Distributed Processing Symposium, (IPDPS'03)*, IEEE Computer Society Press, 2003.
8. C. Banini, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, No.4, pp. 319-330, April 2004.
9. B. Veeravalli, X. Li, and C.C. Ko, "On the influence of start-up costs in scheduling divisible loads on bus networks", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 11, No. 12, pp. 1288-1305, December 2000.
10. S. Suresh, V. Mani, and S.N. Omkar, "The effect of start-up delays in scheduling divisible loads on bus networks: An alternate approach", *Computers and Mathematics with Applications*, Vol. 40, pp. 1545-1557, 2003.
11. M. Drozdowski and P. Wolniewicz, "Multi-Installment Divisible job processing with communication start-up cost", *Foundations of Computing and Decision Sciences*, Vol.27, No.1, pp.43-57, 2002.
12. V. Bharadwaj, H. F. Li and T. Radhakrishnan, "Scheduling divisible loads in bus network with arbitary release time," *Computers and Mathematics with Applications*, vol. 32, no. 7, pp. 57-77, 1996.
13. V. Bharadwaj and Wong Han Min, "Scheduling Divisible Loads on Heterogeneous Linear Daisy Chain Networks with Arbitrary Processor Release Times," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, No. 3, pp. 273-288, 2004.
14. V. Bharadwaj and Wong Han Min, "Scheduling Divisible Loads on Heterogeneous Linear Daisy Chain Networks with Arbitrary Processor Release Times," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, No. 3, pp. 273-288, 2004.
15. H.J. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.
16. D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, New York, 1989.
17. L. David, *Handbook of Genetic Algorithms*, Van Nostrand Reingold, New York, 1991.
18. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, AI Series, Springer-Verlag, New York, 1994.
19. F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," *Artificial Intelligence Review*, vol. 12, No. 4, pp. 265-319, 1998.
20. K. Deb, *Multi-objective optimization using evolutionary algorithms*, Wiley, Chichester, 2001.
21. F. Herrera, M. Lozano, and A. M. Sanchez, "Hybrid crossover operators for real-coded genetic algorithms: An experimental study," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 2002.
22. F. Herrera, M. Lozano, A.M. Snchez, "A Taxonomy for the Crossover Operator for Real-Coded Genetic Algorithms: An Experimental Study," *International Journal of Intelligent Systems* vol. 18, pp. 309-338, 2003.
23. C.R. Houck, J.A. Joines, and M.G. Kay, "A genetic algorithm for function optimization: A Matlab implementation", *ACM Transactions on Mathematical Software*, vol.22, pp.1-14, 1996.
24. V. Mani, S. Suresh, and H.J. Kim, " real-coded Genetic Algorithms for Optimal Static Load Balancing in Distributed Computing System with Communication Delays", Lecture Notes in Computer Science, LNCS 3483, pp. 269-279, 2005.

25. F. Herrera, M. Lozano, "Gradual Distributed Real-Coded Genetic Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 43-63, 2000.
26. T. P. Hong, H. S. Wang, "Automatically Adjusting Crossover Ratios of Multiple Crossover Operators," *Journal of Information Science and Engineering*, vol. 14, no. 2, pp. 369-390, 1998.
27. T. P. Hong, H. S. Wang, W. Y. Lin, W. Y. Lee, "Evolution of Appropriate Crossover and Mutation Operators in a Genetic Process," *Applied Intelligence*, vol. 16, pp. 7-17, 2002.
28. H. S. Yoon, B. R. Moon, "An Empirical Study on the Synergy of Multiple Crossover Operators," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 212-223, 2002.